Escuela
Politécnica
Superior

Clasificador de dibujos hecho a mano

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Baptiste Samper de Diego
Tutor/es:
Miguel Ángel Cazorla Quevedo
Francisco Gómez Donoso

Julio 2020

Universitat d'Alacant
Universidad de Alicante

Clasificador de dibujos hecho a mano

Autor
Baptiste Samper de Diego

Tutor/es
Miguel Ángel Cazorla Quevedo
Departamento de Ciencia de la Computación e Inteligencia Artificial
Francisco Gómez Donoso
Departamento de Ciencia de la Computación e Inteligencia Artificial

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2020

Preámbulo

En este proyecto se realiza un trabajo de investigación sobre la mejor forma de solucionar un problema de clasificación, sobre dibujos hechos a mano en menos de 30 segundos. Gracias a esta investigación y sus conclusiones, se pueden trasladar a otros problemas de características similares, como reconocimiento de firmas.

Agradecimientos

Me gustaría agradecer a todas las personas que me han ayudado a realizar el trabajo de fin de grado y me han soportado durante todo el proceso, aunque en ocasiones no me he aguantado ni yo.
Me gustaría agradecer a mis tutores, Miguel y Fran, que me han ayudado en los momentos de dificultad y me han dado la motivación para seguir.
También me gustaría agradecer a mi familia y amigos por toda la ayuda prestada y la paciencia de estar ahí desde el principio.
A todos vosotros muchas gracias, esto es por vosotros.

Una computadora puede ser llamada inteligente si logra engañar a una persona haciéndole creer que es un humano.
Alan Mathison Turing

Índice general

Índice de figuras

1 Introducción

El reconocimiento de dibujos o imágenes hechas a mano es una rama de la automatización a explotar y mejorar. Además de ayudar a los ingenieros a comprender y investigar sobre el Aprendizaje Automático o *Machine Learning*. En este caso se creó un juego para este propósito que se llama [QuickDraw] (s.f.) colaborador con [Google] (2020) . Además de aparecer en [Kaggle] (2020) página de retos relacionados con *Deep Learning* y Redes Neuronales. Con este tipo retos, instan a los ingenieros que den la mejor solución para resolver el problema[1].

1.1 El problema a resolver

Nosotros nos centraremos en resolver un problema de clasificación de imágenes. Gracias al uso de una red neuronal. Más concretamente, un clasificador que clasifique 34 millones de imágenes(dibujos hechos a mano) clasificadas en 340 clases diferentes.

1.2 ¿Porqué es importante resolver este problema?

Resolver este tipo de problemas es importante. Puesto que el reconocimiento y clasificación de imágenes, es una rama muy importante de la inteligencia artificial y una área del conocimiento informático muy interesante. La cual se aplica hoy en día en numerosas ocasiones. Por ejemplo, en el reconocimiento facial en los dispositivos móviles.

1.3 Posibles soluciones del problema

La mayoría de soluciones para este problema actualmente, se basan en el uso de una Red Neuronal para el Aprendizaje Automático. Esta red neuronal es un clasificador y una [CNN] (2020) (Convolutional Neural Network) o Red Neuronal Convolucional. La cual se basa en aprender patrones de imágenes y clasificar las imágenes en diferentes clases, de forma automática.

1.4 La solución propuesta del problema

Por nuestra parte se propone una solución CNN, pero con una arquitectura de red que está siendo en este momento el estado del arte. Hablamos de la [Efficientnet] (2020) que demuestra

[1] `https://www.kaggle.com/c/quickdraw-doodle-recognition/overview`

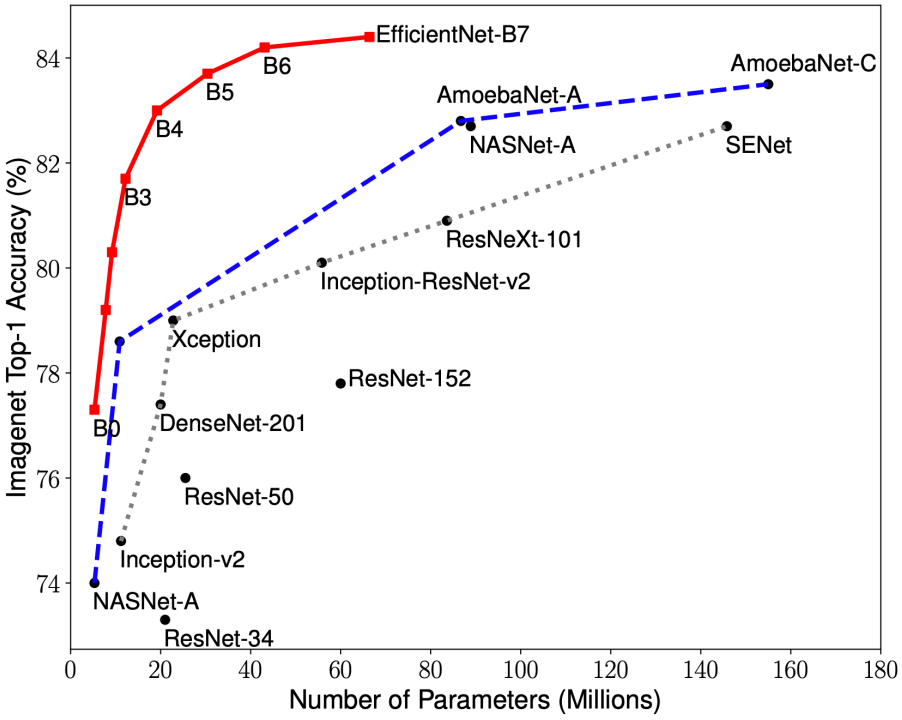


Figura 1.1: Comparación de tamaño con las diferentes redes actualmente.

un acierto superior a sus antecesores, teniendo menos parámetros como se puede apreciar en la Figura 1.1:

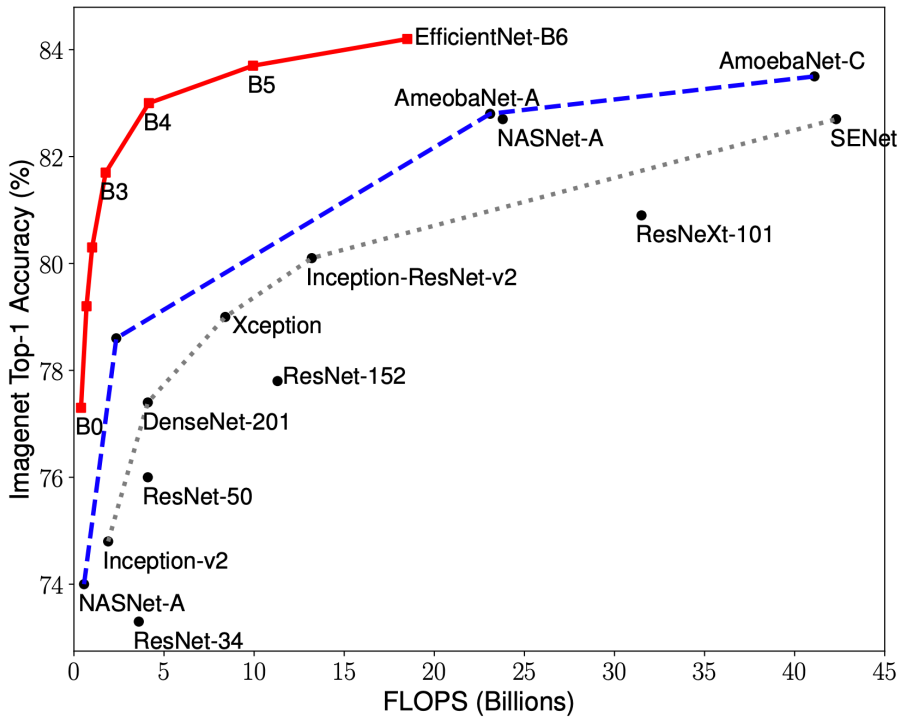


Figura 1.2: Comparación de los FLOPs con las diferentes redes actualmente.

1.5 Estructura de tfg

Una vez sabiendo esto, se explicará cómo está estructurado el documento para facilitar su navegación:

- En el capítulo 2 se realiza una introducción teórica de las redes CNN así como las diferentes Efficientnet (el actual estado del arte) que existen.
- En el capítulo 3 se explican los objetivos generales y específicos del proyecto.
- En el capítulo 4 se explica el planteamiento que se ha seguido en el desarollo y se explican las herramientas y tecnologías que se han utilizado.
- En el capítulo 5 se relata detalladamente el proceso de desarrollo, el preprocesamiento de datos, equilibrado de clases, la implementación de la Efficientnet, los problemas encontrados y las soluciones propuestas.
- En el capítulo 6 se centra en documentar todo el proceso de experimentación y los resultados obtenidos.
- En el capítulo 7 se hace una valoración del proyecto y se proponen más posibles soluciones.

2 Marco Teórico

En este capítulo se realiza una introducción teórica de las redes CNN así como las diferentes Efficientnet (el actual estado del arte) que existen.

Pero primero se explicará unos conceptos que se necesitan saber para poder entender los conceptos más avanzados.

2.1 Conceptos básicos necesarios

El concepto más básico del que partimos es *Machine Learning* o Aprendizaje Automático. Ya que en este concepto se encapsulan las tecnologías que se emplean en las redes neuronales. Esto es imprescindible para entender la estructura de aprendizaje que se va a utilizar. La Efficientnet.

2.1.1 Machine Learning

El [aprendizaje automático] (2020) o aprendizaje de máquinas (del inglés, machine learning) es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. Se dice que un agente aprende cuando su desempeño mejora con la experiencia; es decir, cuando la habilidad no estaba presente en su genotipo o rasgos de nacimiento. De forma más concreta, los investigadores del aprendizaje de máquinas buscan algoritmos y heurísticas para convertir muestras de datos en programas de computadora, sin tener que escribir los últimos explícitamente. Los modelos o programas resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio (potencialmente infinito) de datos.

Además de que el aprendizaje automático también está estrechamente relacionado con el reconocimiento de patrones. Esto es lo que verdaderamente nos interesa puesto que con el reconocimiento de patrones en imágenes. Seremos capaces de automatizar un clasificador de imágenes, gracias a una red neuronal. Este es nuestro siguiente concepto básico.

2.1.2 Redes Neuronales

El segundo concepto básico derivado de *Machine Learning* que debemos saber es el de las [redes neuronales] (2020), las cuales consisten en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida,

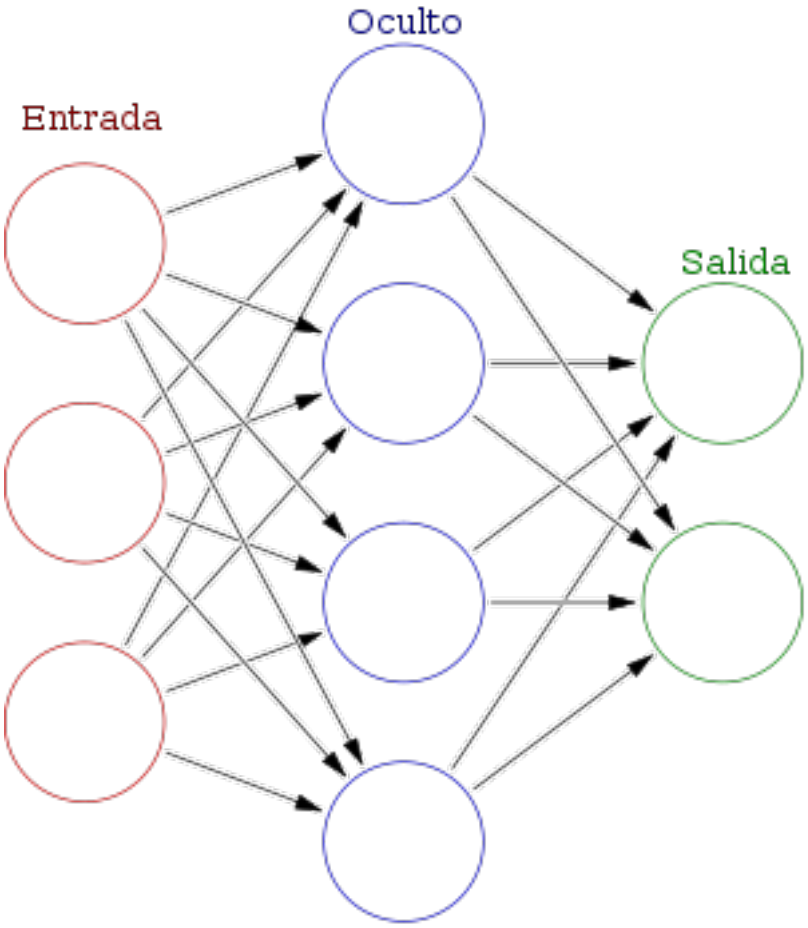


Figura 2.1: Estructura de una red neuronal.

como se puede apreciar en la Figura 2.1.

Además cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Esta función se conoce como función de activación.

Las redes neuronales aprenden y se forman a sí mismas, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Para realizar este aprendizaje automático, normalmente, se intenta minimizar una función de pérdida que evalúa la red en su total. Los valores de los pesos de las neuronas se van actualizando buscando reducir el valor de la función de pérdida. Este proceso se realiza mediante la propagación hacia atrás.

Figura 2.2: Uno de los usos de las redes neuronales es reconocimiento de voz.

Las redes neuronales se han utilizado para resolver una amplia variedad de tareas. Como la visión por computador y el reconocimiento de voz como se puede apreciar en la Figura 2.2. Que son difíciles de resolver usando la ordinaria programación basado en reglas. Nosotros concretamente la usaremos para poder clasificar imágenes.

2.1.2.1 Redes Neuronales Convolucionales

Las [redes neuronales convolucionales] (2020) es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales (imágenes), son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones. Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.

Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo como se aprecia en la Figura 2.5. Al final de la red se encuentran neuronas de perceptrón sencillas para realizar la clasificación final sobre las características extraídas. La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales como se muestra en la Figura 2.4 y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas como se aprecia en la Figura 2.3.

En la fase de extracción de características, las neuronas sencillas de un perceptrón son reemplazadas por procesadores en matriz que realizan una operación sobre los datos de imagen 2D que pasan por ellas, en lugar de un único valor numérico.

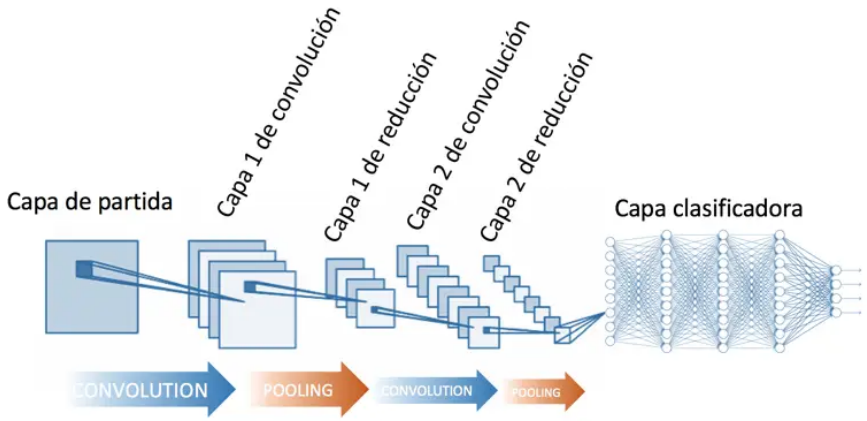


Figura 2.3: Estructura de una red neuronal convolucional.

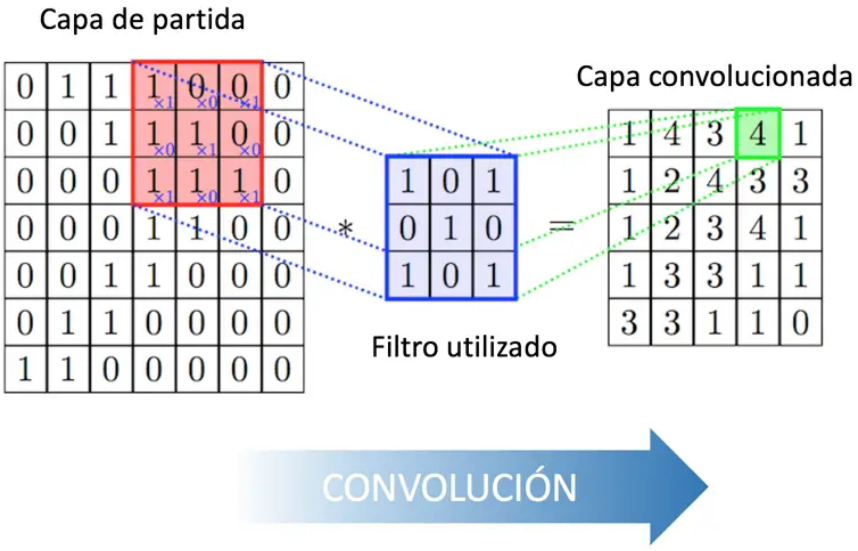


Figura 2.4: Filtro convolucional.

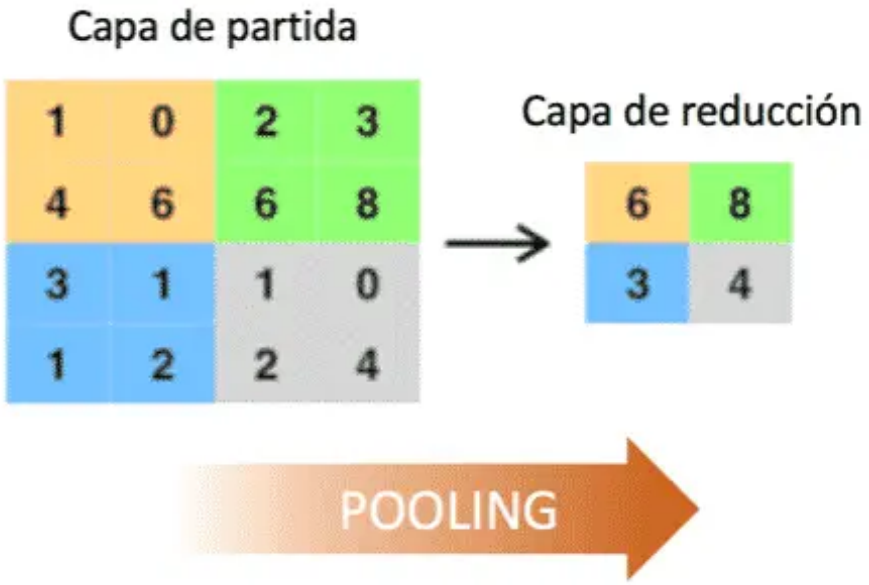


Figura 2.5: Reducción o pooling.

Gracias a esto entendemos cómo funciona un clasificador de imágenes, hecho con una red neuronal convolucional. Uno de los usos, es por ejemplo, que aprenda a distinguir números hechos a mano como se aprecia en la Figura 2.6. En nuestro caso se encarga de clasificar entre más de 340 clases, pero esto lo detallaremos más adelante.

2.2 Soluciones propuestas por otras personas para el mismo problema

Podemos encontrar soluciones propuestas por otras personas a nuestro problema en la página de Kaggle[1]. La gente suele proponer diferentes estrategias para solucionar el problema. Aunque se suele optar por transformar las coordenadas de los puntos en imágenes, para poder entrenar una CNN. Además de probar diferentes arquitecturas tales como, resnet18, resnet34, resnet50, resnet101, resnet152, resnext50, resnext101, densenet121, densenet201, vgg11, pnasnet, incresnet, polynet, nasnetmobile, senet154, seresnet50, seresnext50, seresnext101. Aunque existen diferentes opciones, de solucionar el problema. Como el uso de una red recurrente que vaya recorriendo la sucesión de puntos.

Después de investigar diferentes arquitecturas de redes. Decidimos quedarnos con la EfficientNet, puesto que era el estado del arte de ese momento (como se explicó en la introducción), es decir, es la arquitectura que se espera que de mejores resultados.

[1] `https://www.kaggle.com/c/quickdraw-doodle-recognition/discussion/73738`

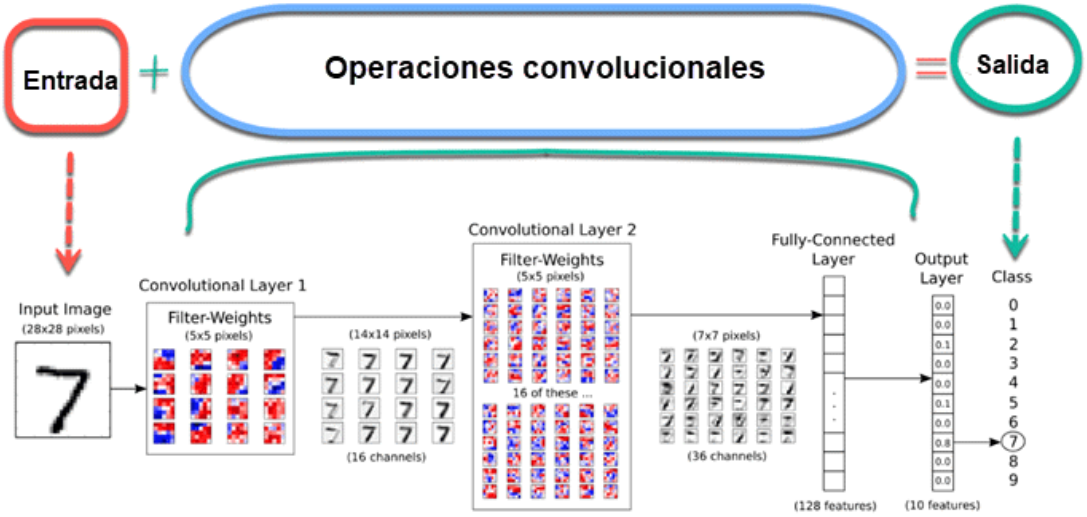


Figura 2.6: Ejemplo de aplicación de red neuronal convolucional.

3 Objetivos

El objetivo principal de este proyecto es encontrar una buena solución para el problema de el reconocimiento y clasificación de imagenes de dibujos hechos a mano, propuesto por Kaggle[1]

Para llevar a cabo esto, se tendrán que hacer los siguientes hitos u objetivos:

- Investigar sobre las soluciones ya dadas sobre este problema o problemas similares.
- Analizar el dataset proporcionado por la página de kaggle.
- Establecer el entorno para poder realizar las pruebas pertinentes.
- Investigar sobre el estado del arte y probar esa solución.
- Probar diferentes soluciones y probar la mejor configuración de la red.
- Analizar los resultados obtenidos.

[1] `https://www.kaggle.com/c/quickdraw-doodle-recognition/data`

4 Metodología

En este capítulo explicaremos el proceso de desarrollo del trabajo, en el primer punto explicaremos los pasos seguidos y a continuación las herramientas utilizadas.

4.1 Pasos del proceso de desarrollo

1. Estudio del problema planteado:

   En este punto se analiza el problema a resolver, como qué dataset nos proporcionan, de que está compuesto, de cuantos ejemplos disponemos.

2. Análisis y adaptación del dataset para optimizarlo:

   Balanceo de clases, depuración de ejemplos mal clasificados, conversión de sucesión de puntos a imágenes, para poder procesarlas.

3. Estudio de las diferentes estructuras de las redes:

   Estudiar las diferentes redes que existen y utilizar la que promete mejor solución, para problemas similares al nuestro (clasificación de imágenes).

4. Implementación de la red:

   En nuestro caso la implementación de Efficientnet, y adaptándolo a nuestro problema en concreto.

5. Entrenamiento de la red y valoración de los resultados:

   Finalmente entrenar la red a clasificar correctamente y valorar los resultados obtenidos mediante gráficas.

4.2 Herramientas y librerías utilizadas durante el proceso de desarrollo

Aquí explicamos las herramientas que hemos utilizado para el desarrollo del trabajo, como el control de versiones, lenguaje, librerías, sistema operativo.

4.2.1 Ubuntu 18.04.04 LTS

Ubuntu[1] es el sistema operativo que hemos elegido para realizar el trabajo, puesto que es software libre y hay una gran comunidad que lo mantiene y mejora además de una extensa documentación.

[1] `https://releases.ubuntu.com/18.04/`

4.2.2 Sublimetext

[Sublimetext] (2020) como editor para desarrollar todo el código que necesitábamos para llevar a cabo el proyecto.

4.2.3 Python 3.6.9

[Python] (2020) es un lenguaje de programación de alto nivel, muy utilizado en la industria del software, sobretodo a la hora de trabajar con análisis de datos y con redes neuronales.

4.2.4 Git

[Git] (2020) es una herramienta que facilita el control de versiones del software, esto ayuda especialmente a tener un control del proceso de producción de código, a poder cambiar a versiones anteriores.

4.2.5 TensorFlow

[TensorFlow] (2020) es una plataforma de código abierto de extremo a extremo para el aprendizaje automático, TensorFlow facilita la creación de modelos de aprendizaje automático tanto para principiantes como expertos. Que además soporta a más alto nivel, la herramienta de Keras, librería para desarrollar modelos de aprendizaje automático. En nuestro caso usamos tensorflow-gpu en la versión 2.0.0

4.2.6 Keras

[Keras] (2020) es una API diseñada para seres humanos, no para máquinas. Keras sigue las mejores prácticas para reducir la carga cognitiva: ofrece API consistentes y simples, minimiza la cantidad de acciones del usuario requeridas para casos de uso comunes y proporciona mensajes de error claros y procesables. También cuenta con una amplia documentación y guías para desarrolladores. En nuestro caso usamos keras en la versión 2.2.4

4.2.7 OpenCV

[OpenCV] (2020) es una librería de código abierto, para la manipulación de imágenes, escrita en C++. En nuestro caso la usamos para crear las imágenes de entrenamiento, a partir de los puntos que forman el dibujo del dataset y poder visualizarlas.

4.2.8 Matplotlib

[Matplotlib] (2020) es una librería de creación de gráficas y visualización de datos, que es compatible con python. En nuestro caso la usamos para hacer las diferentes gráficas que

hemos necesitado para visualizar los distintos datos, tales como, el número de muestras del dataset, el estado del entrenamiento.

4.2.9 H5py

[H5pt] (2020) es una librería para almacenar de forma comprimida y organizada una gran cantidad de información. En nuestro caso la usamos para organizar nuestro inmenso dataset (34 millones de imágenes) en dos conjuntos, el 70 porciento para el conjunto de train y el 30 porciento para el conjunto de test.

4.2.10 NumPy

[NumPy] (2020) es una librería que facilita la manipulación de arrays. En nuestro caso la utilizamos para almacenar los datos de entrenamiento en este tipo de estructura.

5 Desarrollo

En esta parte del trabajo explicaremos cómo se han llevado a cabo los diferentes pasos para finalizar el proyecto y el porqué de las decisiones tomadas durante el proceso.

5.1 Primeros pasos para solucionar el problema

Lo primero para solucionar un problema es entenderlo y para ello, nos enfocaremos en entender nuesto problema a resolver[1]. A primera vista observamos que la idea principal del problema. Es que proporcionemos una solución que clasifique imágenes. Para ello la solución que proponemos es la de crear una red neuronal que clasifique esas imágenes.

5.1.1 DataSet

Ahora profundizamos más en el dataset que nos proporcionan desde la web. Se puede observar que el dataset consta de 50 millones de dibujos categorizados entre 340 clases diferentes. En nuestro caso elegimos el train simplifield puesto que contiene solo la información relevante. En el dataset se aprecia que en vez de imágenes, son las coordenadas de la sucesión de puntos que conforman el dibujo.

5.1.2 El porqué del uso de una CNN

Aunque pueda parecer contraproducente, el hecho de tener que generar las imágenes a partir de los puntos. En vez de trabajar directamente con los puntos. Tiene su motivo. Puesto que al usar una CNN, no se tiene en cuenta la variable temporal. Por otro lado, si la entrenamos a partir de los puntos, esto tiene una variable temporal. Puesto que un punto va antes que otro(esto no nos interesa puesto que cada persona hacer el dibujo de forma distinta).

5.1.3 Modificando el DataSet

Al analizar detenidamente el dataset con maś profundidad. El dataset consta de muestras que están validadas y que no están validadas. Que una muestra esté validada significa que cuya categorización ha sido comprobada por un humano. Cómo se muestra en la Figura 5.1 se puede observar la cantidad de muestras válidadas y en la Figura 5.2 se puede apreciar la cantidad de muestras no validadas. En los experimentos solo están involucradas las muestras

[1] `https://www.kaggle.com/c/quickdraw-doodle-recognition/overview`

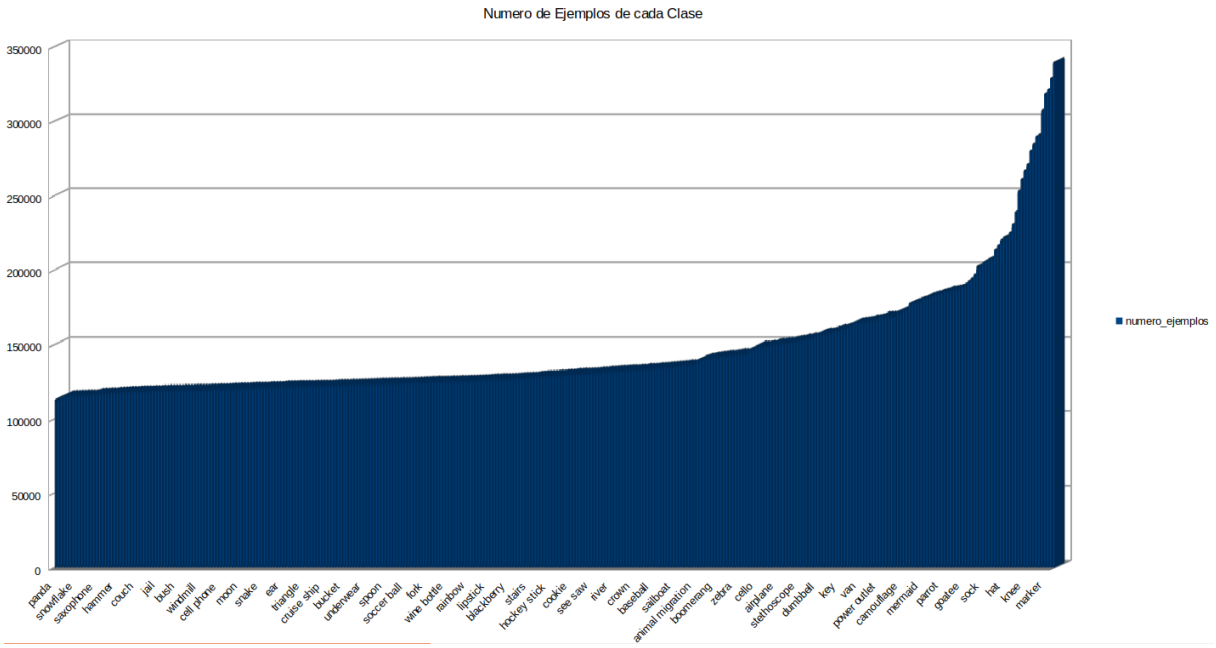


Figura 5.1: Podemos observar la cantidad de dibujos por cada clase del dataset inicial.

válidadas.

Para aclarar, las Figuras 5.1 y 5.2. En el eje X aparecen el nombre de las 340 clases como se muestra en la Figura 5.3

Gracias a estas gráficas podemos observar que no todas las clases tiene el mismo número de elementos, así que procedemos ha hacer un equilibrado de clases para que la red aprenda con la mayor calidad posible.

En nuestro caso nos quedamos con 100.000 ejemplos de cada clase (menos los que tienen un poco menos de ejemplos, que nos quedamos con todos ellos).

Una vez ”limpiado” el dataset. Creamos un programa en Python y OpenCV. Para convertir las coordenadas en una imagen en blanco y negro(1 solo canal) de tamaño de 256 por 256 píxeles, con el resultado que se puede apreciar en la Figura 5.4.

Al crear los primeros dibujos a partir de las sucesiones de puntos. Se apreciaba una lentitud exasperante (calculando más de 8 meses, algo impensable). El cálculo se hizo a partir de medir el tiempo de creación de una imágen y la multiplicación por las 34 millones de imágenes que había que crear. En consecuencia la solución que propusimos fue paralelizar el programa gracias a usar la librería threads. En nuestro caso el portátil tiene 8 núcleos. Así que tardamos 1 mes las 24 horas el portátil encendido generando las imágenes.

En este punto tenemos un total de 34 millones de imágenes divididas en 340 clases (100.000

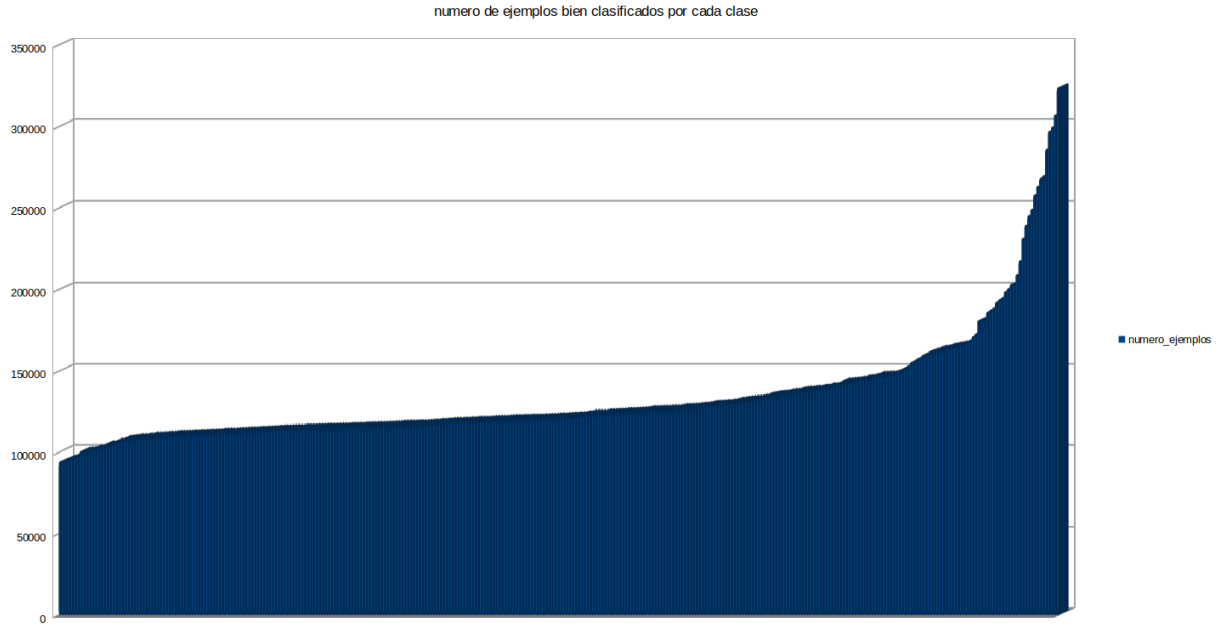
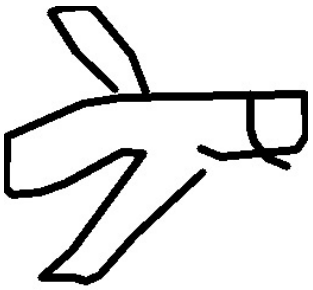


Figura 5.2: Podemos observar la cantidad de dibujos por cada clase del dataset una vez filtrado por los ejemplos válidos.

```
['airplane', 'alarm clock', 'ambulance', 'angel', 'animal migration', 'ant', 'anvil', 'apple', 'arm', 'asparagus', 'axe', 'backpack', 'banana',
'bandage', 'barn', 'baseball bat', 'basket', 'basketball', 'bat', 'bathtub', 'beach', 'bear', 'beard', 'bed', 'bee', 'belt', 'bench', 'bicycle', 'binoculars',
'bird', 'birthday cake', 'blackberry', 'book', 'boomerang', 'bottlecap', 'bowtie', 'bracelet', 'brain', 'bread', 'bridge', 'broccoli', 'broom', 'bucket',
'bulldozer', 'bus', 'bush', 'butterfly', 'cactus', 'calculator', 'calendar', 'camel', 'camera', 'camouflage', 'campfire', 'candle', 'cannon', 'canoe', 'car',
'carrot', 'castle', 'cat', 'ceiling fan', 'cell phone', 'cello', 'baseball', 'blueberry', 'cake', 'chair', 'crown', 'elbow', 'flip flops', 'hammer',
'house plant', 'lightning', 'motorbike', 'paintbrush', 'pillow', 'rake', 'sheep', 'speedboat', 'string bean', 'The Great Wall of China', 'trumpet',
'chandelier', 'church', 'circle', 'clarinet', 'clock', 'cloud', 'coffee cup', 'compass', 'computer', 'cookie', 'cooler', 'couch', 'cow', 'crab',
'crayon', 'crocodile', 'cruise ship', 'cup', 'diamond', 'dishwasher', 'diving board', 'dog', 'dolphin', 'donut', 'door', 'dragon', 'dresser', 'drill',
'drums', 'duck', 'dumbbell', 'ear', 'elephant', 'envelope', 'eraser', 'eye', 'eyeglasses', 'face', 'fan', 'feather', 'fence', 'finger', 'fire hydrant',
'fireplace', 'firetruck', 'fish', 'flamingo', 'flashlight', 'floor lamp', 'flower', 'flying saucer', 'foot', 'fork', 'frog', 'frying pan',
'garden hose', 'garden', 'giraffe', 'goatee', 'golf club', 'grapes', 'grass', 'guitar', 'hamburger', 'hand', 'harp', 'hat', 'headphones', 'hedgehog',
'helicopter', 'helmet', 'hexagon', 'hockey puck', 'hockey stick', 'horse', 'hospital', 'hot air balloon', 'hot dog', 'hot tub', 'hourglass', 'house',
'hurricane', 'ice cream', 'jacket', 'jail', 'kangaroo', 'key', 'keyboard', 'knee', 'ladder', 'lantern', 'laptop', 'leaf', 'leg', 'light bulb', 'lighthouse',
'line', 'lion', 'lipstick', 'lobster', 'lollipop', 'mailbox', 'map', 'marker', 'matches', 'megaphone', 'mermaid', 'microphone', 'microwave', 'monkey',
'moon', 'mosquito', 'mountain', 'mouse', 'moustache', 'mouth', 'mug', 'mushroom', 'nail', 'necklace', 'nose', 'ocean', 'octagon', 'octopus', 'onion',
'oven', 'owl', 'paint can', 'palm tree', 'panda', 'pants', 'paper clip', 'parachute', 'parrot', 'passport', 'peanut', 'pear', 'peas', 'pencil', 'penguin',
'piano', 'pickup truck', 'picture frame', 'pig', 'pineapple', 'pizza', 'pliers', 'police car', 'pond', 'pool', 'popsicle', 'postcard', 'potato',
'power outlet', 'purse', 'rabbit', 'raccoon', 'radio', 'rain', 'rainbow', 'remote control', 'rhinoceros', 'river', 'roller coaster', 'rollerskates',
'sailboat', 'sandwich', 'saw', 'saxophone', 'school bus', 'scissors', 'scorpion', 'screwdriver', 'sea turtle', 'see saw', 'shark', 'shoe', 'shorts',
'shovel', 'sink', 'skateboard', 'skull', 'skyscraper', 'sleeping bag', 'smiley face', 'snail', 'snake', 'snorkel', 'snowflake', 'snowman', 'soccer ball',
'sock', 'spider', 'spoon', 'spreadsheet', 'square', 'squiggle', 'squirrel', 'stairs', 'star', 'steak', 'stereo', 'stethoscope', 'stitches', 'stop sign',
'stove', 'strawberry', 'streetlight', 'submarine', 'suitcase', 'sun', 'swan', 'sweater', 'swing set', 'sword', 't-shirt', 'table', 'teapot', 'teddy-bear',
'telephone', 'television', 'tennis racquet', 'tent', 'The Eiffel Tower', 'The Mona Lisa', 'tiger', 'toaster', 'toe', 'toilet', 'tooth', 'toothbrush',
'toothpaste', 'tornado', 'tractor', 'traffic light', 'train', 'tree', 'triangle', 'trombone', 'truck', 'umbrella', 'underwear', 'van', 'vase', 'violin',
'washing machine', 'watermelon', 'waterslide', 'whale', 'wheel', 'windmill', 'wine bottle', 'wine glass', 'wristwatch', 'yoga', 'zebra', 'zigzag']
```

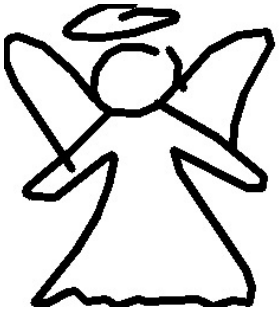
Figura 5.3: Nombre de las 340 clases.



(a) Ejemplo de dibujo airplane.

(b) Ejemplo de dibujo clock.

(c) Ejemplo de dibujo ambulance.



(d) Ejemplo de dibujo angel.

Figura 5.4: Podemos observar el dibujo convertido en imágen que la red ya puede procesar.

ejemplos cada clase). Que pesan en total 850 GB. Al ver tal cantidad de datos, la opción que vimos más viable. Fue agrupar el dataset en un tipo de archivo que se utiliza mucho a la hora de organizar un dataset. Este tipo de extensión es .hdf5. La cual, te permite tener archivos de forma comprimida y además te permite organizarlos de forma que puedes realizar operaciones sobre los datos de forma sencilla.

Por tanto, el paso siguiente es realizar el programa que se encarga de realizar ese documento, pero ocurría el mismo problema que al crear las imágenes. Que tardaba demasiado. Así que decidimos paralelizar con los 8 núcleos y creamos un .hdf5 por cada clase un total de 340 .hdf5.

5.2 Organizando el DataSet

Estos ficheros los organizamos de la siguiente manera. Un 70 porciento de las imágenes de cada clase para entrenar la red(con la etiqueta datostrain) además de su correspondiente etiqueta en formato onehot para que la red pueda usarlo(con la etiqueta labeltrain) y un 30 porciento de las imágenes de cada clase para comprobar cómo está aprendido la red(con la etiqueta test) además de su correspondiente etiqueta en formato onehot para que la red pueda usarlo(con la etiqueta labeltest). Todo este proceso tardó otro mes y acabó pesando todos los archivos .hdf5 320 GB. Algo que valía la pena ya que hacía los datos mucho más manejables.

5.3 Implementando la red

Una vez, teniendo el dataset preparado para que la red pueda entrenar. Se debe implementar la red para que pueda usar los datos. En nuestro caso, el primer paso es investigar el estado del arte. Es decir, cuál es la arquitectura de red que en este momento está dando mejores resultados para problemas similares al nuestro (clasificación de imágenes).

Nosotros encontramos la Efficientnet la cual daba los mejores resultados. Por otro lado, es una red que aunque optimizada, no deja de ser una red con muchos parámetros y a la hora de entrenar ocupa mucho en memoria.

Así que el siguiente paso, era implementar la red. Una vez hecho el programa con las funciones más comunes de Keras. Surgió el siguiente problema, no podíamos cargar todas las imágenes de entrenamiento a la vez en memoria para entrenar. En consecuencia, propusimos la siguiente solución. Hacer nuestro propio flujo de entrenamiento. Es decir, hacer a mano el entrenamiento y controlar en todo el momento con cuantas imágenes entrena(batch) nuestra ren en cada iteración de cada época.

Más concretamente se utiliza una técnica que consiste en hacer dos arrays de índices. Uno para el de train y otro para el test. En nuestro caso, es un array de 340 (por el número de clases que tenemos) elementos en los cuales cada elemento es un array de 70.000 en el caso de train y 30.000 en caso de test (por que es el número de imágenes de cada clase).

Una vez, tenemos los arrays de índices. Los barajamos(shuffle) de forma aleatoria. Para que la red no aprenda alguna característica que perjudique a la red. Puesto que, si se entrena a una red siempre de la misma forma (el mismo orden en cada época) puede ocurrir lo mencionando anteriormente. De esta forma aseguramos un buen entrenamiento.

A continuación, usamos los arrays de índices, para cargar en memoria tantas imágenes como podamos antes de que se llene la memoria. En nuestro caso, como se nos echaba el tiempo encima, decidimos entrenar en un servidor. El cual disponía de 1 tarjeta 2080Ti de 11GB. La cual, nos ahorraría bastante tiempo a la hora de entrenar.

Pero cuando empezamos a entrenar, vimos que la arquitectura de la Efficientnet ocupa bastante en memoria. Así que, solo podíamos entrenar con pocas imágenes a la vez. Más concretamente, 50 imágenes cada iteración(entendemos por iteración un entrenamiento con el tamaño de batch de 50 imágenes, cuando se hagan suficientes iteraciones para ver todas las imágenes una vez diremos que hemos concluido una época).

Sin embargo, en consequencia de tener que entrenar con tantas imágenes(34 millones). Pues no disponíamos de tanto tiempo para entrenar. Así que, vamos recogiendo datos de entrenamiento y probando la red con el conjunto de test. Cada 1.000 iteraciones para ver el progreso de la red.

6 Resultados

A continuación analizaremos los diferentes resultados que hemos ido obteniendo al ir entrenando nuestra red neuronal.

Cabe recalcar que al entrenar con tanto volumen de imágenes(34 millones), el entrenamiento lleva mucho tiempo. Por tanto, nosotros por falta de tiempo no hemos podido entrenar tanto como quisiéramos. Pero aún así, tenemos suficientes datos como para poder mostrar unas gráficas y analizarlas.

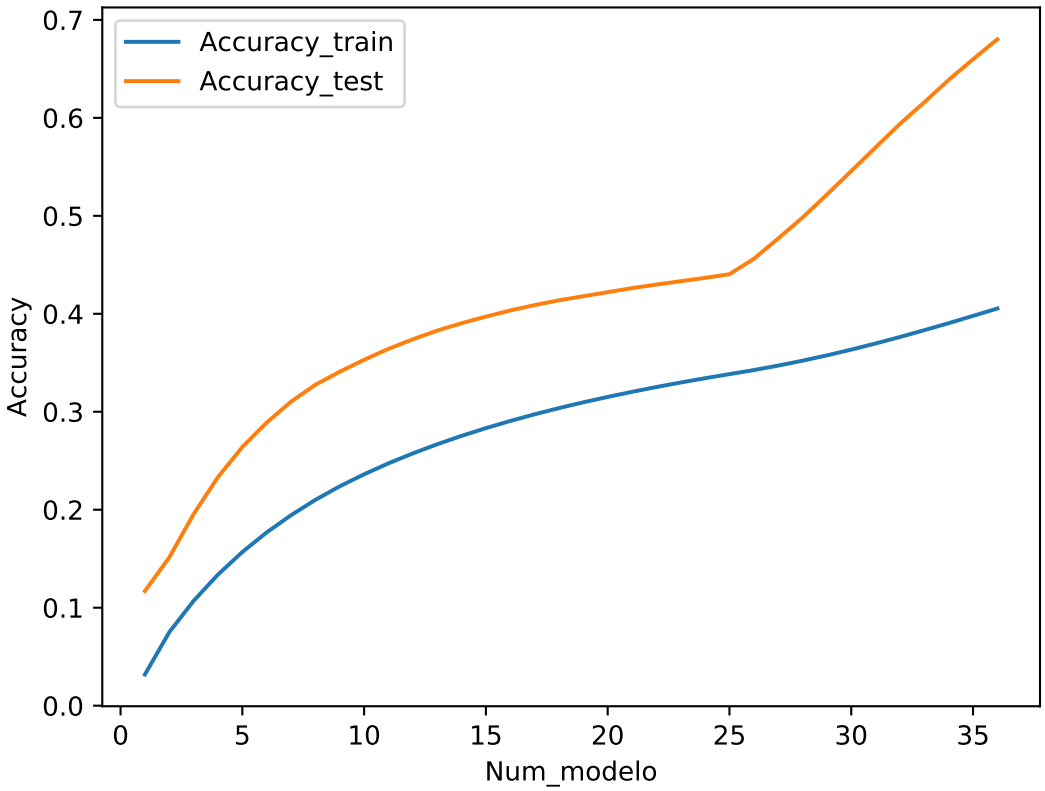


Figura 6.1: Podemos observar la precisión de nuestro modelo en el ejeY, respecto al ejeX el número de modelo, es el número(en miles) de imágenes que ha visto la red, es decir, al ser 35, la red ha entrenado con 35.000 imágenes.

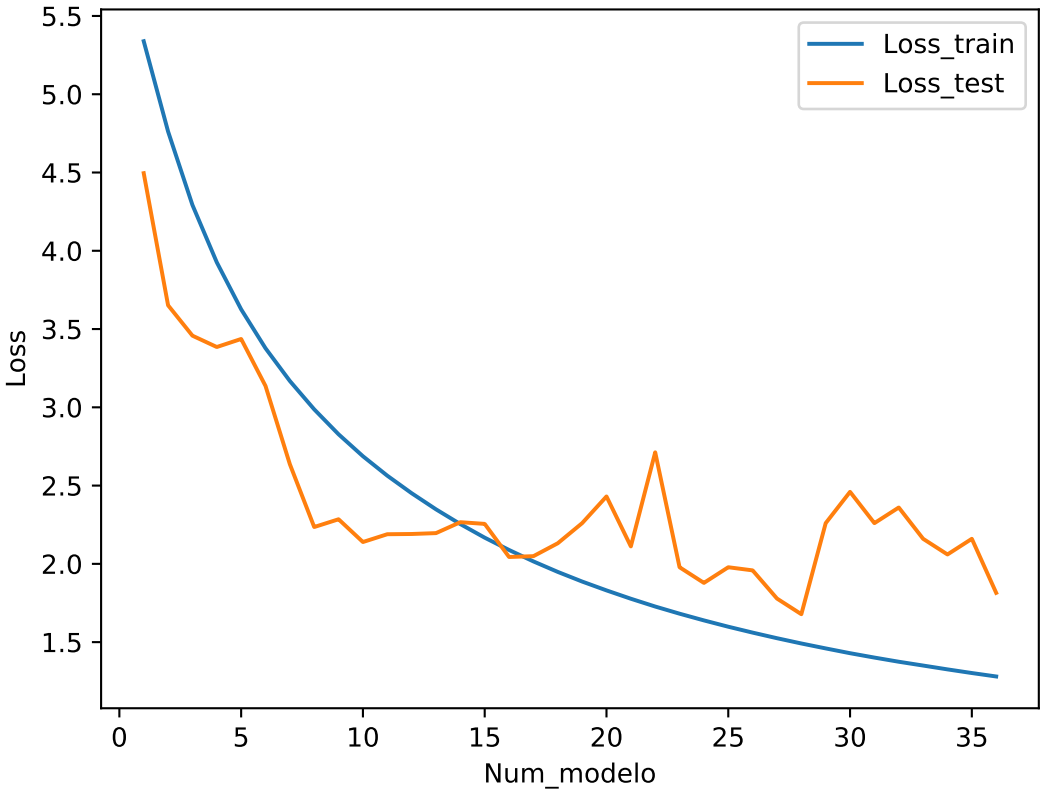


Figura 6.2: Podemos observar la pérdida de nuestro modelo en el ejeY, respecto al ejeX el número de modelo, es el número(en miles) de imágenes que ha visto la red, es decir, al ser 35, la red ha entrenado con 35.000 imágenes.

Analizando los resultados de la Figura 6.1. Nos damos cuenta de que la red sigue mejorando. Es decir, que la red debería seguir entrenando para alcanzar su máximo potencial y más porcentaje de acierto.

A parte de analizar la tendencia general. Podemos fijarnos que el acierto es superior en el conjunto de test. Debido a que a la hora de hacer el test. Se le pasan las mismas imágenes, pero en diferente orden. Puesto que el conjunto de test no varía. Ya que son 30.000 imágenes por cada clase.

Reitero que nosotros por falta de tiempo no hemos podido seguir entrenando nuestro modelo, aunque se intuye una tendencia a seguir mejorando.

Por otro lado, si analizamos la Figura 6.2. Podemos observar lo siguiente. La pérdida de train se mantiene en un constante decrecimiento. Puesto que la solución que da la red respecto a la esperada. Cada vez, se van acercando más.

Sin embargo, la pérdida de test, se manifiesta una tendencia a la baja. Pero con saltos irregulares. Esto se puede deber a que la red todavía no tiene los pesos bien ajustados.

7 Conclusiones

Cerrando este proyecto, dedicamos esta parte a evaluar el trabajo realizado y las diferentes opciones con las cuales se podría seguir con el proyecto y mejorarlo.

Respecto a los resultados del proyecto, se ha podido observar que la arquitectura de red Efficientnet, funciona bien como clasificador de imágenes además de conseguir los siguientes objetivos:

- Realizar un estudio sobre el estado del arte de las redes neuronales.
- Analizar un dataset y modificarlo para poder hacerlo óptimo y funcional.
- Aprender a construir un entorno de trabajo, para poder entrenar una red neuronal.
- Aprender a analizar los resultados obtenidos para obtener conclusiones coherentes.

Además de aprender el uso de las diferentes librerías, tales como OpenCv, keras y demás. Que antes se desconocían.

Por otro lado, las posibles mejoras que se podrían haber llevado a cabo son:

- Continuar entrenando la red neuronal hasta ver unos resultados óptimos, además de probar en vez de una red convolucional, trabajar directamente con la sucesión de puntos con una red neuronal recurrente, para comparar los resultados y ver cuál se adapta mejor al problema.
- Estudiar un posible uso alternativo de los modelos ya entrenados, para reconocer otro tipo de imágenes.

Bibliografía

[aprendizaje automático]. (2020). https://en.wikipedia.org/wiki/Machine_learning.

[CNN]. (2020). https://en.wikipedia.org/wiki/Convolutional_neural_network.

[cómo configurar tu propio flujo de entrenamiento en Keras]. (2020). https://keras.io/guides/writing_a_training_loop_from_scratch/.

[cómo entrenar con grandes cantidades de datos]. (2020). https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/.

[cómo funciona por dentro el método fit de Keras]. (2020). https://keras.io/guides/customizing_what_happens_in_fit/.

[cómo instalar Cuda Toolkit 10.0]. (2020). https://developer.nvidia.com/cuda-10.0-download-archive?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1804&target_type=runfilelocal.

[cómo instalar en Ubuntu todo el setup para poder entrenar la red neuronal]. (2020). https://www.pyimagesearch.com/2019/01/30/ubuntu-18-04-install-tensorflow-and-keras-for-deep-learning/.

[cómo instalar TensorFlow-gpu]. (2020). https://www.tensorflow.org/install/gpu.

[cómo usar un modelo entrenado previamente en Keras]. (2020). https://towardsdatascience.com/step-by-step-guide-to-using-pretrained-models-in-keras-c9097b647b29.

[Efficientnet]. (2020). https://github.com/qubvel/efficientnet.

[ejemplo de cómo se entrena una red neuronal en Keras]. (2020). https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5.

[ejemplo de FineTuning en Keras]. (2020). https://www.learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/.

[Git]. (2020). https://github.com/.

[Google]. (2020). https://experiments.withgoogle.com/collection/ai.

[H5pt]. (2020). https://www.h5py.org/.

[Kaggle]. (2020). https://www.kaggle.com/.

[Keras]. (2020). https://www.keras.io/.

[las diferentes Efficeintnet que existen en Keras]. (2020). https://keras.io/api/applications/efficientnet/#efficientnetb0-function.

[Matplotlib]. (2020). https://www.matplotlib.org/.

[NumPy]. (2020). https://www.numpy.org/.

[OpenCV]. (2020). https://www.opencv.org/.

[Python]. (2020). https://www.python.org/.

[QuickDraw]. (s.f.). https://quickdraw.withgoogle.com/.

[redes neuronales]. (2020). https://es.wikipedia.org/wiki/Red_neuronal_artificial.

[redes neuronales convolucionales]. (2020). https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales.

[repositorio de comparaciones de arquitecturas de red 1]. (2020). https://github.com/keras-team/keras-applications.

[repositorio de comparaciones de arquitecturas de red 2]. (2020). https://github.com/qubvel/efficientnet.

[repositorio de comparaciones de arquitecturas de red 3]. (2020). https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet.

[repositorio de Efficentnet]. (2020). https://github.com/qubvel/efficientnet/blob/master/examples/inference_example.ipynb.

[Sublimetext]. (2020). https://www.sublimetext.com/.

[TensorFlow]. (2020). https://www.tensorflow.org/.